

Distributed RDataFrame: supported backends, latest improvements and future plans

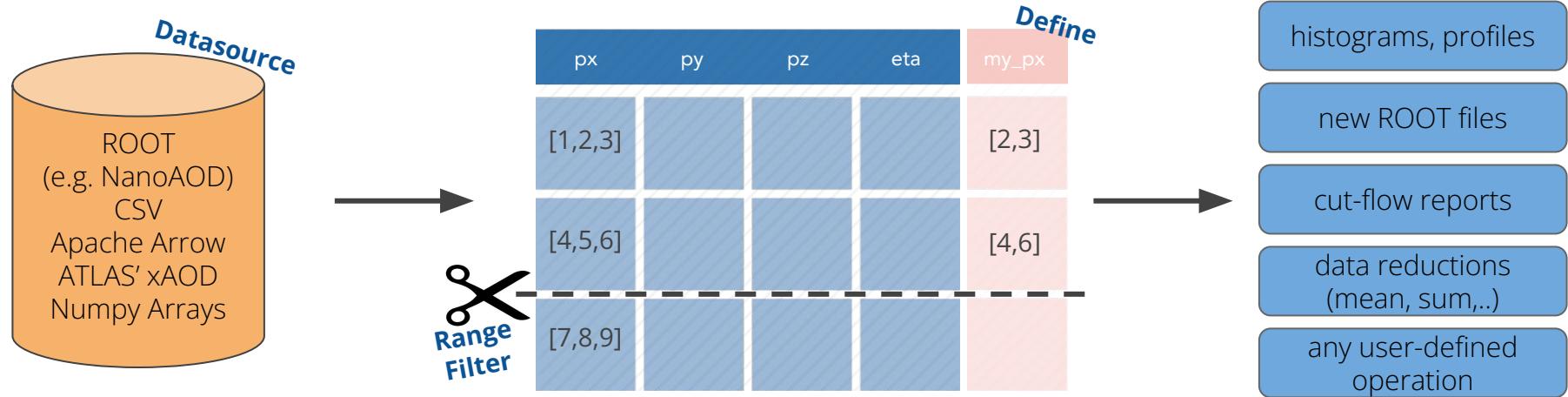
Vincenzo Eduardo Padulano for the ROOT team

ROOT
Workshop
2022

ROOT
Data Analysis Framework
<https://root.cern>



RDataFrame analysis interface



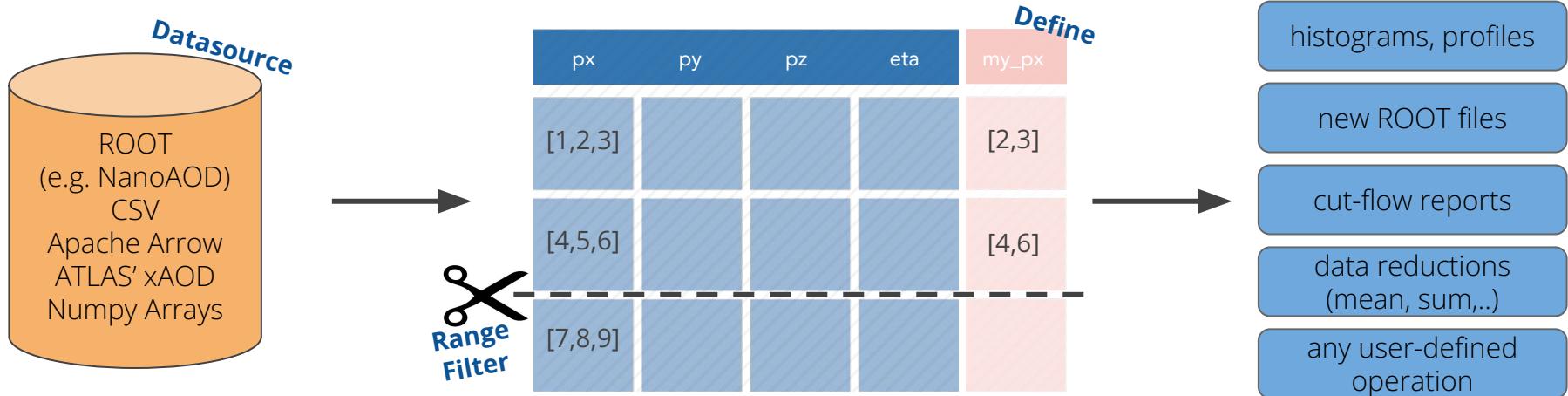
```
# enable multi-threading  
ROOT.EnableImplicitMT()  
df = ROOT.RDataFrame(dataset)
```

```
df = df.Range(2)  
.Define("my_px", "px[eta > 0]")
```

```
# filled in a single pass  
h1 = df.Histo1D("my_px", "w")  
h2 = df.Histo1D("px", "w")
```



RDataFrame analysis interface



Goal

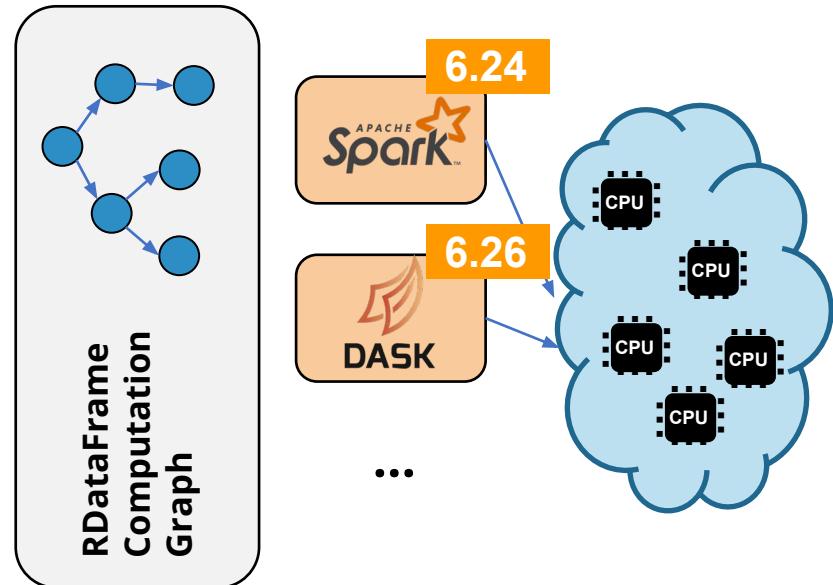
Best performance and ease of use across the board,
for most (all?) HEP analysis use cases

[RDataFrame reference guide](#)

[RDataFrame tutorials](#)

Going Distributed

- Enable **interactive large-scale distributed** data analysis with RDataFrame
- Can run with different schedulers
 - **Spark**
 - **Dask**
 - ...
- Analysis from start to end in a **single interface**





Programming Model

Local

```
from ROOT import RDataFrame
```

Importing RDataFrame

```
df = RDataFrame('treename', 'filename.root')
```

Constructing RDataFrame

```
df2 = df.Filter(...).Define(...)  
h1 = df2.Histo1D(...)  
h1.Draw()
```

Rest of application

Distributed

```
import ROOT  
RDataFrame = \  
ROOT.RDF.Experimental.Distributed.Dask.RDataFrame
```

```
from dask.distributed import Client  
df = RDataFrame('treename', 'filename.root',  
daskclient = Client('tcp://hostname:port'))
```



Programming Model

Local

```
from ROOT import RDataFrame
```

99% of the application stays exactly the same!

Constructing RDataFrame

```
df2 = df.Filter(...).Define(...)  
h1 = df2.Histo1D(...)  
h1.Draw()
```

Rest of application

Distributed

```
import ROOT
```

```
Distributed.Dask.RDataFrame
```

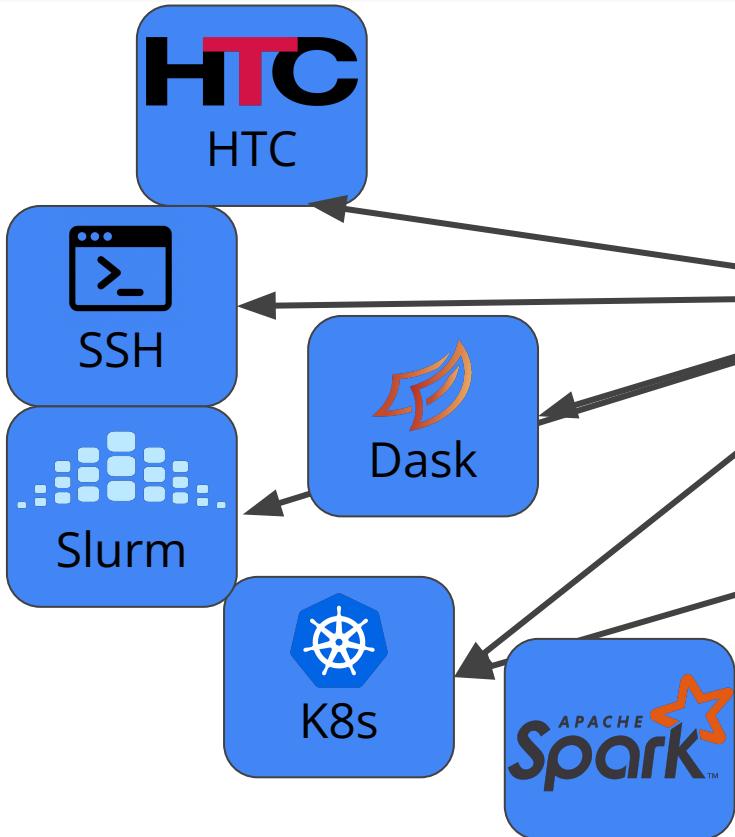
```
import Client
```

```
df = RDataFrame('treename', 'filename.root',  
                daskclient = Client('tcp://hostname:port'))
```





One API, Many Backends



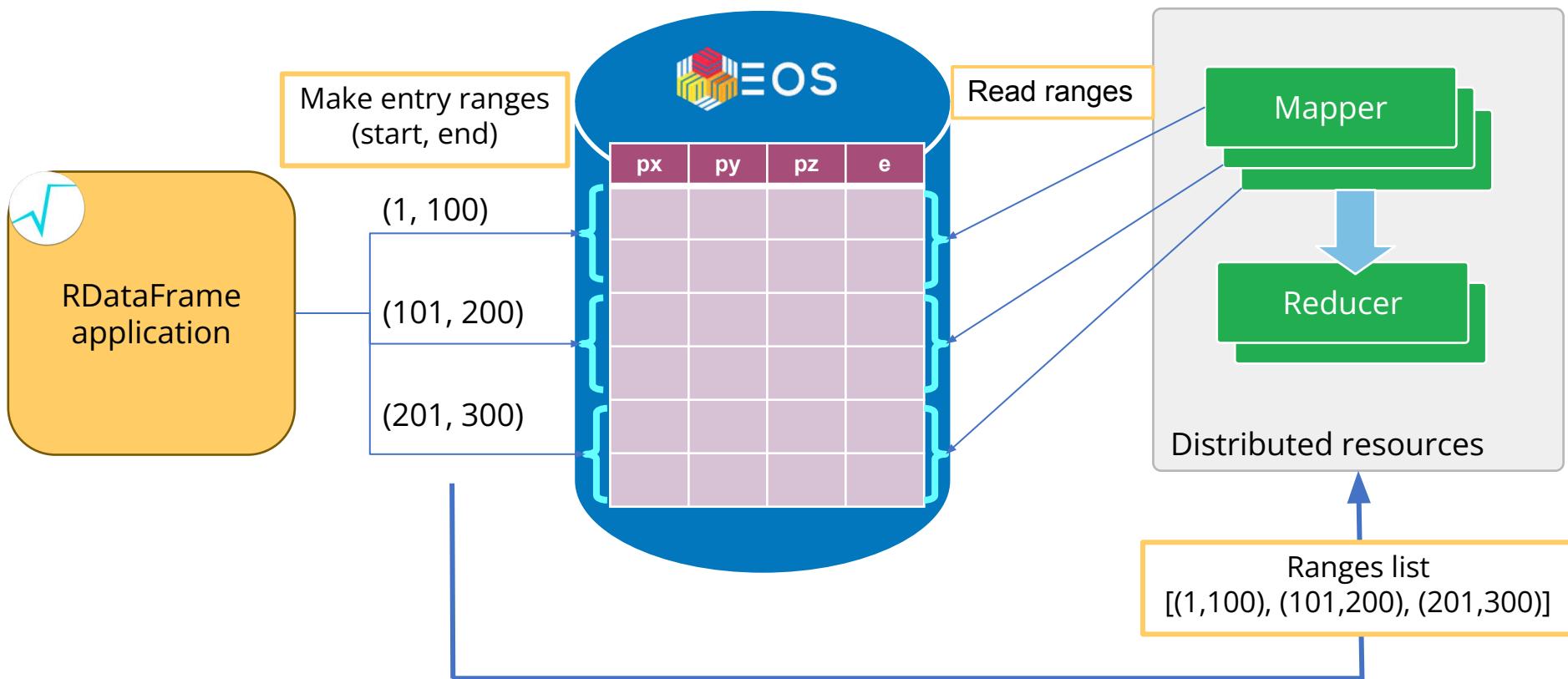
```
from dask.distributed import Client  
df = RDataFrame('treename', 'filename.root',  
                 daskclient = Client('tcp://hostname:port'))
```

```
from pyspark import SparkContext  
df = RDataFrame('treename', 'filename.root',  
                sparkcontext = SparkContext('spark://IP:PORT'))
```

[Demo notebooks with Spark and Dask](#)



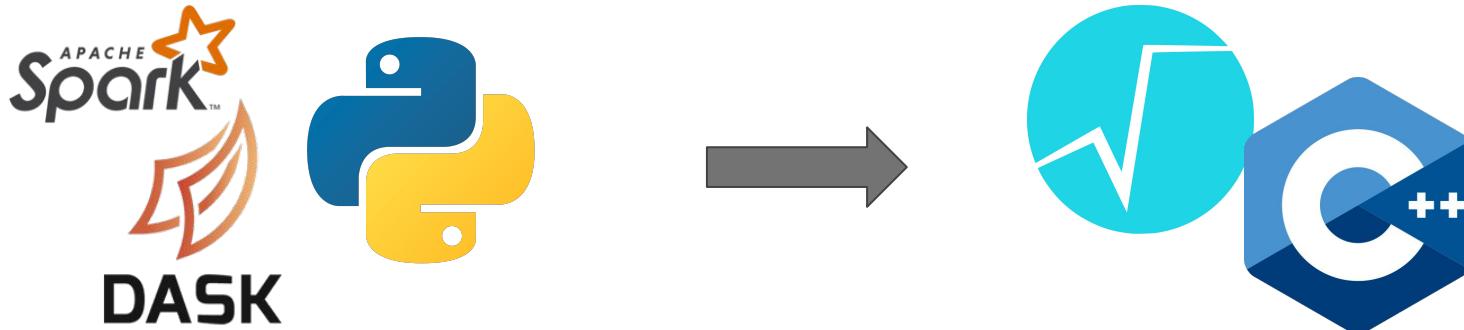
Behind distributed RDataFrame





Distributed C++

- ▶ Python API, C++ event loop
- ▶ Leveraging full power of ROOT I/O
- ▶ Rely on distributed execution engines for scheduling and resource management





Supported RDataFrame API

Almost all of RDF methods are supported in distributed mode:

- AsNumpy
- Count
- Define
- DefinePerSample
- Fill
- Filter
- Graph
- Histo[1,2,3]D
- HistoND
- Max
- Mean
- Min
- Profile[1,2,3]D
- Redefine
- RunGraphs
- Snapshot
- Sum
- Vary
- VariationsFor



Distributing user code: Initialize tasks

`ROOT.RDF.Experimental.Distributed.initialize`

- ▶ Define your own function
- ▶ Runs once at the beginning of each distributed task
- ▶ For example nice if you want to declare C++ code quickly

```
import ROOT
def f():
    ROOT.gInterpreter.Declare('''
        #ifndef MYFUN
        #define MYFUN
        int myfun(){ return 42; }
        #endif
    ''')
```

Using header guards avoids
redefinitions in the workers

```
ROOT.RDF.Experimental.Distributed.initialize(f)
```



Distributing User Code: headers, libraries

Improved interface for user code distribution:

- ▶ Distribute C++ headers or libraries
- ▶ Distribute code snippets (e.g. a “distributed” gInterpreter.Declare with automatic header guards)

Coming Soon

```
ROOT...distribute_headers('myheader.hpp',  
restrict_to_df=mydf)
```

```
ROOT...distribute_libraries('libtest.so',  
restrict_to_df=mydf)
```

```
ROOT...distribute_code('''  
int myfun(){ return 42; }  
''')
```



More Pythonic RDF operations

Injecting Python functions in distributed RDF via Numba
(as seen in today's talk about PyROOT)

```
def f(x: int) -> bool:  
    return x > 10
```

Coming Soon

```
df = RDataFrame('treename', 'filename.root', daskclient = Client('MYENDPOINT'))  
df.Filter(f, ['mycol'])
```



More future improvements

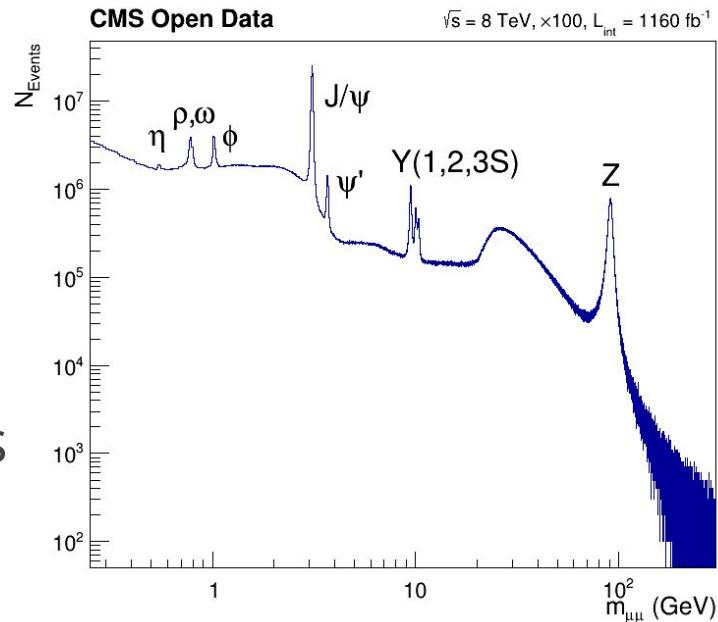
Other future improvements:

- ▶ Automatically pick a good number of partitions for a certain computation graph
- ▶ Better debugging tools: aggregated executions logs from distributed workers



Benchmark example: dataset

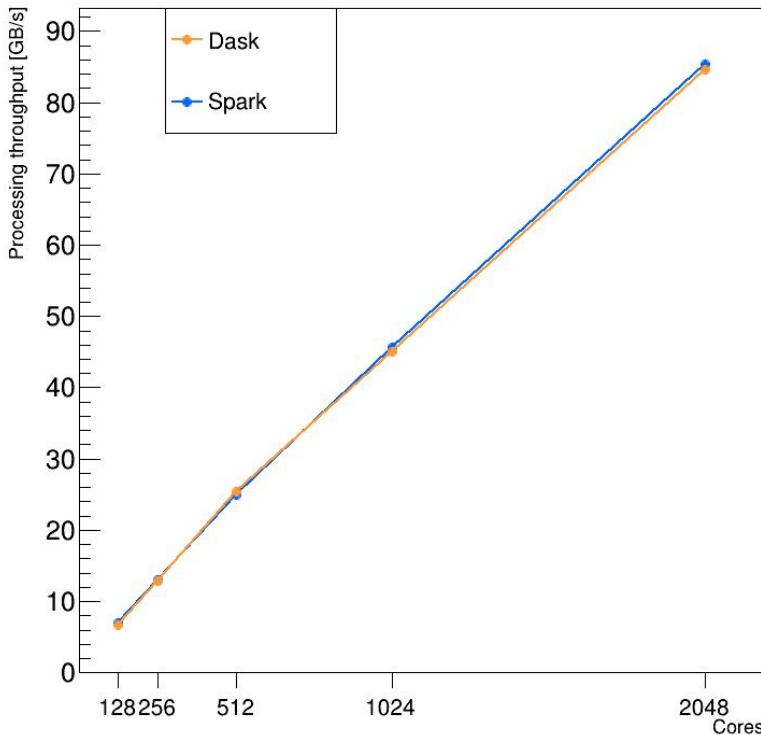
- ▶ Dimuon analysis
- ▶ 4000x original dataset
 - **8800 GB**
 - ZLIB compressed
- ▶ Data is **node-local** during analysis
- ▶ **100%** read and processed



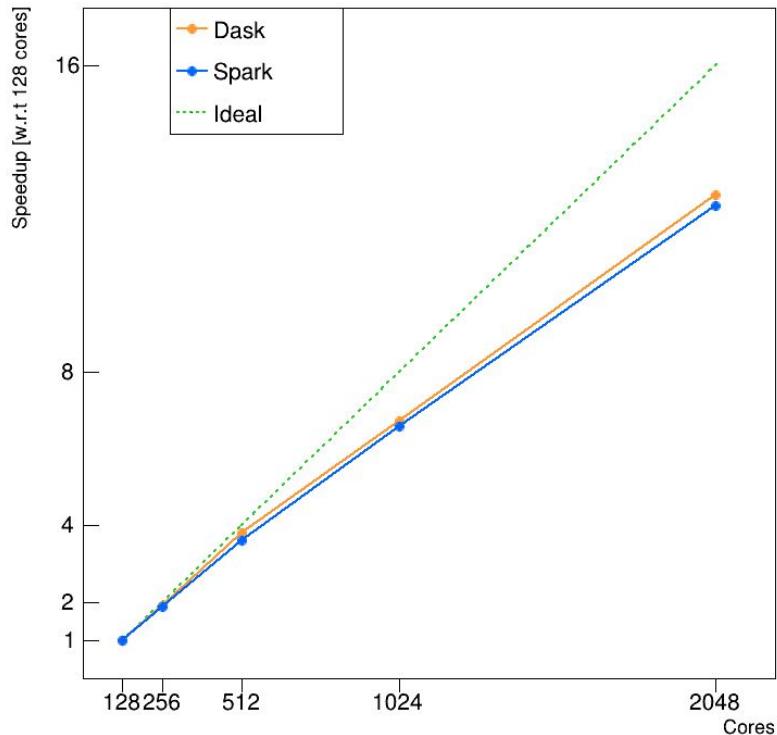


Benchmark example: comparison

processing throughput



speedup



- ▶ **Distributed RDataFrame** is here
 - Requires minimal changes w.r.t. local application
 - Task scheduling with Spark or Dask (and more!)
- ▶ Run interactively in a Python notebook or in a batch system with a Python script
- ▶ **Coming soon:**
 - Simple interfaces to distribute your code
 - More pythonic (distributed) RDataFrame
 - Smoother distributed debugging experience
- ▶ Still experimental and in development, **your feedback is welcome!**